REDIS

Open source in-memory data structure store

Riciputi Jacopo

30 Maggio 2018

Indice

Cos'è

Redis è uno store di strutture dati completamente in-memory, prestante e open source.
È utilizzato principalmente come:

- Database;
- Cache;
- Message broker.

Soprattutto per:

- applicazioni Web;
- videogiochi;
- sistemi IoT.

Tra i punti di forza vi sono **velocità** e la **facilità** con cui permette di gestire varie strutture dati.

Perchè è uno dei più utilizzati

Nel mondo di database NoSQL Redis risulta essere il secondo database NoSQL, alle spalle del colosso MongoDB e in 7 posizione se vengono considerati anche i database relazionali.[1]
Tra i suoi **punti di forza**:

- La struttura chiave-valore permette di gestire più strutture dati;
- Il mantenimento dei dati in memoria ed il linguaggio in cui è scritto (C) lo rendono **estramente veloce** e con tempi di latenza minimi;
- Opzionalmente è permette di aggiungere **persistenza**;
- Supporta le transazioni;
- Fornisce funzionalità per eseguire **Lua scripting**, grazie ad un interprete built-in;
- Offre **high avalability** e supporta la suddivisione dei dati all'interno di una rete **cluster** di macchine.

[1] `https://db-engines.com/en/ranking`

Estramente veloce

Primi passi

Primi Passi

Installare Redis richiede qualche semplice passo se scaricato dalla pagina ufficiale[2] oppure è possibile ottenere una versione completa su Debian o derivate lanciando il comando ***sudo apt-get install redis-server***.

- Una volta installato, con il comando ***redis-server*** viene avviata un'instanza sulla porta di defalut (6379)
- È possibile collegarsi da terminale a questa instanza tramite il client ***redis-cli***

A questo punto si è completamente operativi.
Esempio base:
Redis per la gestione di semplici chiavi fornisce i comamdi **SET** e **GET**

- **SET** richiede una chiave e il corrispettivo valore.
  Es. SET foo bar → OK
- **GET** Data una chiave ne ritorna il valore. Es. GET foo → bar

[2] `https://redis.io/download`

Strutture Hash

HSET consente di creare una chiave a cui associare più coppie chiave-valore.
> HSET hfoo firstKey firstValue
(integer 1)
> HSET hfoo secondKey secondValue
(integer 1)

È possibile riottenere questi valore in differenti modi:

- HGET hfoo firstKey → "firstValue"
- HGETALL hfoo
  1) "firstKey" 2) "firstValue" 3) "secondKey" 4) "secondValue"
- E' possile altrimenti ottenere tutte le chiavi o tutti i valori rispettivamente con i comandi **HKEYS** e **HVALS**

Liste

Redis offre la possibilità di gestire le liste, tramite i comandi **LPUSH** e **LPOP** per le operazioni in testa, **RPUSH** e **RPOP** per quelle in coda.

```
> LPUSH list 2
(integer) 1
> RPUSH list 3
(integer) 2
> LPUSH list 1
(integer) 3
```

Redis restituirà i valori in ordine "1", "2", "3".
È possibile verificarlo con l'istruzione **LRANGE**

```
> LRANGE list 0 -1
1) "1"
2) "2"
3) "3"
```

Set

Nella gestione dei set in Redis si possiedono due comandi principali, **SADD** e **SMEMBERS**, rispettivamente per aggiungere valori ad una chiave e visualizzare i suoi elementi.

```
> SADD myset 1 2 3
(integer) 3
> SMEMBERS myset
1. "3"
2. "1"
3. "2"
```

Decisamente più interessanti i **Sordet Set**.

Sorted Set

Questa struttura dati è gestita da Redis associando ad ogni valore uno *score*.
L'ordinamento viene infatti eseguito in base al valore di questo *score*.
Se per due valori coincide sarà eseguito sul valore lessicografico di quest'ultimi.
In questo caso i comandi base sono:

- **ZADD**: Prende in ingresso chiave, score e valore e li immaggazzina
- **ZRANGE**: Data una chiave, indice di partenza e di fine ne visualizza in ordine **crescente** i valori;
- **ZREVRANGE**: Data una chiave, indice di partenza e di fine ne visualizza in ordine **decrescente** i valori;
- **ZRANGEBYLEX**, **ZREVRANGEBYLEX**, **ZREMRANGEBYLEX** e **ZLEXCOUNT** per manipolarli in base al valore lessicografico.

Sorted Set

```
> ZADD classe 1996 "Mario Rossi"
(integer) 1
> ZADD classe 1993 "Fabio Bianchi"
(integer) 1
> ZADD classe 1987 "Roberto Verdi"
(integer) 1
> ZADD classe 1990 "Carlo Neri"
(integer) 1

> ZRANGE classe 0 -1
1) "Roberto Verdi"
2) "Carlo Neri"
3) "Fabio Bianchi"
4) "Mario Rossi"
```

Nodejs client

npm install ioredis - Client molto semplice per Nodejs

```
var Redis = require("ioredis");
var redis = new Redis({
  port: 6379,
  host: "127.0.0.1",
  family: 4,
});

redis.set("foo", "bar");
redis.get("foo", function(err, result) {
  console.log(result);
});

redis.del("foo");
redis.get("foo", function(err, result) {
  console.log(result);
});
```

Nodejs client

Output:

```
bar
null
```

Qualcosa di più "complicato":

```
redis.hmset("me",
            "name","jacopo",
            "surname", "riciputi",
            "age", 22);

redis.hgetall("me", function(err, result){
    console.log(result);
});
```

```
Output:
{ name: 'jacopo', surname: 'riciputi', age: '22'}
```

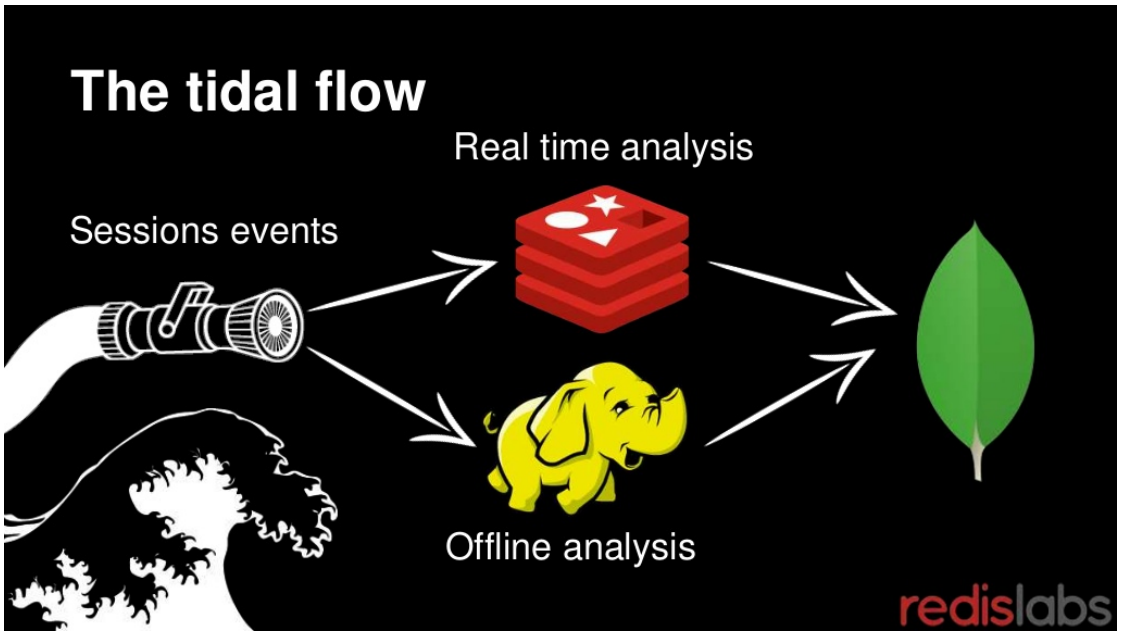
Utilizzo

Caching

Problemi:

- Difficoltà nella digestione dei dati;
- Mobile, IoT, web 2.0 producono continuamente informazioni;
- Impossibile eseguire processi real-time.

Soluzione → Pulire i dati con Redis!!



Caching

L'idea è di accompagnare la sessione dell'utente con Redis.
Al suo termine i dati saranno processati, "ripuliti" e passati a contenitori più grandi (es. MongoDB)
Le sessioni terminano dopo un logout o un timeout.

- **Facile** identificare il logout
- **Difficile** rilevare un timeout, soprattutto considerando le migliaia di sessioni che possono essere attive

Qui Redis viene in soccorso!
Permette infatti di associare una scadenza alle chiavi e di gestirne le notifiche.

Message Broker

Redis offre la possibilità di gestire un **PUB/SUB messaging paradigm**. Sarà possibile perciò client iscritti ad un determinato canale ed publisher che inviano messaggi. Ogni emittente non è programmato per inviare ad uno specifico destinatario ma a tutti i subscriber di un determinato canale.

- Iscrizione con **SUBSRIBE foo**
- Publicazione con **PUBLISH foo hello**
- È possibile sottoscriversi utilizzando i pattern con **PSUBSCRIBE h?llo**
  Accetterà messaggi da hello, hallo, hxllo ma non da hllo.

Database

Il mondo odierno necessita di analisi istantanee, i tradizionali tool necessitano ore, se non giorni!
Redis offre:

- Analisi in tempo reale;
- Transazioni atomiche;
- Supporto ad un elevatissimo numero di transazioni per secondo;
- L'assenza di struttura e la semplicità nella modellazione dei dati lo rende un valido elemento come base di un backend web.
- È possibile scalare le dimensioni dell'applicazione aggiungendo macchine in una rete cluster.

Casi d'uso

Clash of Clans

- Gioco real-time con milioni di utenti.
- Il datastorage principale è MongoDB.
- Utilizza Redis per il salvataggio e la progressione delle partite.
- Nel dettaglio per l'aggiornamento delle risorse real-time, i punteggi e le classifiche.

Waze

- Focus sul traffico real-time
- Più di 10 milioni di utenti collegati nelle ore di punta
- Database principale: MongoDB.
- Utilizza Redis per gli aggiornamenti in tempo reale sul traffico, i veicoli e i passeggeri.
  I dati venono appoggiati al suo interno per poi periodicamente elaborati per **aggiustamenti** e **notifiche**.

Conclusioni

In sintesi

- Prestazioni e facilità d'uso;
- Molteplici strutture dati;
- Scalabilità;
- Forte community, driver per molti linguaggi;
- Vari scenari: Caching, Message Broker, Database;
- Open source ed estendibile: RediSearch / rediSQL / ReJSON / Redis Graph;
- Cuore italiano ! [3]

[3] https://github.com/antirez